

Project 1 - Part 1

Zhicheng Zhang - G45149856

1. Introduction

Get Start with [SimpleScalar](#).

2. Implementation

Prepare

Environment

- Docker image [krlmlr/debian-ssh](#) on Debian 10 (host).
- File `~/singlesim-3v0e.tgz` is downloaded from <http://www.singlescalar.com/>.
- File `~/benchmarks.tar.gz` is downloaded from <http://www.ecs.umass.edu/ece/koren/architecture/Singlescalar/benchmarks.tar.gz>.

Script

```
# install
apt-get install tar build-essential

# unzip
tar zxvf singlesim-3v0e.tgz
tar zxvf benchmarks.tar.gz

# path
PATH=$PATH:~/singlesim-3.0
```

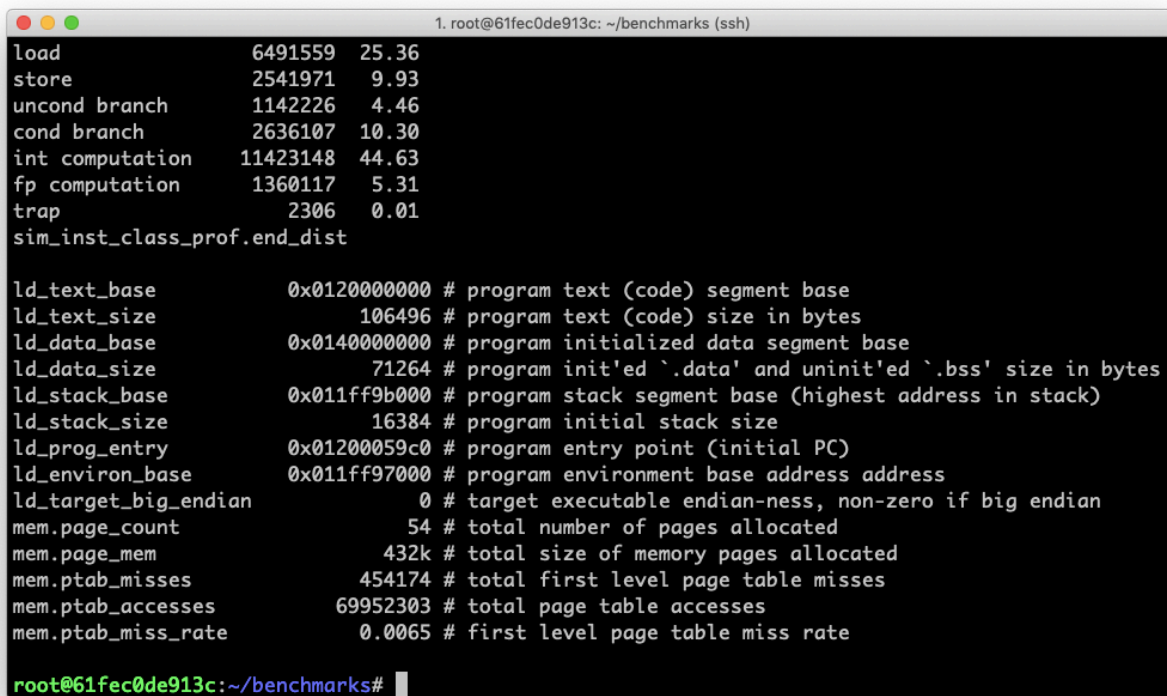
Compile

```
cd ~/singlesim-3.0
make clean
make config-alpha
make
make sim-tests
cd ~
```

Run

```
cd ~/benchmarks
sim-profile -iclass anagram.alpha words < anagram.in > anagram.out
sim-profile -iclass go.alpha 50 9 2stone9.in > 2stone9.out
sim-profile -iclass compress95.alpha < compress95.in > compress95.out
sim-profile -iclass cc1.alpha -O 1stmt.i > 1stmt.s
cd ~
```

3. Result



```
load          6491559  25.36
store         2541971   9.93
uncond branch 1142226   4.46
cond branch   2636107  10.30
int computation 11423148 44.63
fp computation 1360117  5.31
trap           2306    0.01
sim_inst_class_prof.end_dist

ld_text_base    0x0120000000 # program text (code) segment base
ld_text_size    106496 # program text (code) size in bytes
ld_data_base    0x0140000000 # program initialized data segment base
ld_data_size    71264 # program init'ed '.data' and uninit'ed '.bss' size in bytes
ld_stack_base   0x011ff9b000 # program stack segment base (highest address in stack)
ld_stack_size   16384 # program initial stack size
ld_prog_entry   0x01200059c0 # program entry point (initial PC)
ld_enviro_base  0x011ff97000 # program environment base address address
ld_target_big_endian 0 # target executable endian-ness, non-zero if big endian
mem.page_count  54 # total number of pages allocated
mem.page_mem    432k # total size of memory pages allocated
mem.ptab_misses 454174 # total first level page table misses
mem.ptab_accesses 69952303 # total page table accesses
mem.ptab_miss_rate 0.0065 # first level page table miss rate

root@61fec0de913c:~/benchmarks#
```

4. Conclusion

Benchmark	Total # of Instructions	Load %	Store %	Uncond Branch %	Cond Branch %	Integer Computation %	Floating pt Computation %
anagram.alpha	25597435	25.36	9.93	4.46	10.30	44.63	5.31
go.alpha	545811809	30.62	8.17	2.58	10.96	47.64	0.03
compress.alpha	88001	1.54	79.35	0.19	5.67	13.23	0.00
gcc.alpha	337340977	24.67	11.47	4.12	13.33	46.30	0.11

5. Discussion

1. Is the benchmark memory intensive or computation intensive?

It seems that `compress.alpha` is memory intensive and others are computation intensive.

2. Is the benchmark mainly using integer or floating point computations?

These benchmark are mainly using integer computations.

3. What % of the instructions executed are conditional branches? Given this %, how many instructions on average does the processor execute between each pair of conditional branch instructions (do not include the conditional branch instructions)

See the table below:

Benchmark	Total # of Instructions	Cond Branch %	Total # of Cond Branch	Total # of Others	Average # between Cond Branch
anagram.alpha	25597435	10.30	2636107	22961328	8.71
go.alpha	545811809	10.96	59795799	486016010	8.13
compress.alpha	88001	5.67	4993	83008	16.63
gcc.alpha	337340977	13.33	44969464	292371513	6.50